Towards a Flexible Query Language for FSM-Based Databases

Rafik Bouaziz[1] and Salem Chakhar[2]

[1] Dep. of Computer Science, FSEG, University of Sfax,
Route de l'Aroport, BP 1088 3018 Sfax, Tunisia,
raf.bouaziz@fsegs.rnu.tn,
WWW home http://www.fsegs.rnu.tn
[2] LAMSADE, University of Paris Dauphine,
Place du Maréchal de Lattre de Tassigny,
75775 Paris Cedex 16, France,
chakhar@lamsade.dauphine.fr,
WWW home page: http://www.lamsade.dauphine.fr/~chakhar

Abstract. Defining and processing flexible queries is an important research topic in database area. We may distinguish two ways to support flexible querying in databases: (i) developing interface systems that allow queries in pseudo-natural language, or (ii) developing an extended SQL-like languages. In this paper, we have adopted this second approach. The objective of this paper is to introduce a conceptual query language for accessing FSM-based databases. FSM is a data model that has been recently proposed.

Key words: Fuzzy Database, Flexible Querying, Fuzzy Logic

1 Introduction

Defining and processing flexible queries is one of the topics that seem to afford the greatest potential for further developments in database research area [2]. The key idea in flexible querying is to introduce preferences inside queries [2]. This means that the answers to these queries are no longer a crisp set and entities that "partially" much the preferences stated in the query are also provided to the user. We may distinguish two approaches to support flexible querying. The first approach consists in developing interface systems that allow queries in pseudo-natural language (e.g. [8]). The second approach proposes SQL-like languages (e.g. [7]). In this paper, we have adopted this second approach. The general idea of this approach consists in introducing thresholds in the "FROM" and/or "WHERE" clauses to ensure that entities/tuples and/or attribute values verifying these threshold values are provided to the user. Since the attribute values may be fuzzy, conditions expressed in the "WHERE" clause are naturally fuzzy ones.

The objective of this paper is to introduce a conceptual query language for accessing FSM-based databases and illustrate some examples of data retrieve operations. FSM is a fuzzy semantic data model that has been recently proposed

[4][3]. FSM is mapped to a fuzzy relational object database model (FRO) and implemented on PostgreSQL. The proposed query language uses the concepts of perspective class and qualification, introduced by [5], and introduces thresholds in the "FROM" and "WHERE" clauses. The thresholds in the "FROM" clause correspond to the *global degree of membership* (d.o.m) (cf. §2) and may be mapped to the support of fuzziness at entity/class level. The thresholds in the "WHERE clause correspond to the *partial* d.o.m. (cf. §2) and may be mapped to the support of fuzziness at the attribute level.

The paper is structured as follows. Section 2 briefly presents FSM. Section 3 addresses some implementation issues. Section 4 introduces the notions of perspective class and qualification, which are of importance in FSM query formulation. Section 5 deals with query formulation in FSM and provides some illustrative examples of data retrieve operations. Section 6 deals with query processing. Section 7 concludes the paper.

2 Fuzzy Semantic Model

In this section we briefly present FSM. We focalize only on concepts needed to introduce the proposed query language. Details can be found in [4][3]. The *space of entities* E is the set of all entities of the interest domain. A *fuzzy entity* e in E is a natural or artificial entity that one or several of its properties are fuzzy. In other words, a fuzzy entity verifies only (partially) some extent properties (see below) of its class. A *fuzzy class* K in E is a collection of fuzzy entities: $K = \{(e, \mu_K(e)) : e \in E \wedge \mu_K(e) > 0\}$. μ_K is a characteristic or *membership function* and $\mu_K(e)$ represents the *degree of membership* (d.o.m) of the fuzzy entity e in the fuzzy class K. Membership function μ_K maps the elements of E to the range $[0, 1]$ where 0 implies no-membership and 1 implies full membership. A value between 0 and 1 indicates the extent to which entity e can be considered as an element of fuzzy class K. A fuzzy class is a collection of fuzzy entities having some similar properties. Fuzziness is thus induced whenever an entity verifies only (partially) some of these properties. We denote by $X_K = \{p_1, p_2, \cdots, p_n\}$, $n \geq 1$, the set of these properties for a given fuzzy class K. X_K is called the *extent* of fuzzy class K. The extent properties may be derived from the attributes of the class and/or from common semantics. The degree to which each of the extent properties determines fuzzy class K is not the same. Indeed, there are some properties that are more discriminative than others. To ensure this, we associate to each extent property p_i a non-negative weight w_i reflecting its importance in deciding whether or not an entity e is a member of a given fuzzy class K. We also impose that $\sum_{i=1}^{n} w_i > 0$.

On the other hand, an entity may verify fully or partially the extent properties of a given fuzzy class. Let D^i be the basic domain of extent property p_i values and P^i is a subset of D^i, which represents the set of possible values of property p_i. The *partial membership function* of an extent property value is $\rho_{P^i_K}$ which maps elements of D^i into $[0, 1]$. For any attribute value $v_i \in D^i$, $\rho_{P^i_K}(v_i) = 0$ means that fuzzy entity e violates property p_i and $\rho_{P^i_K}(v_i) = 1$

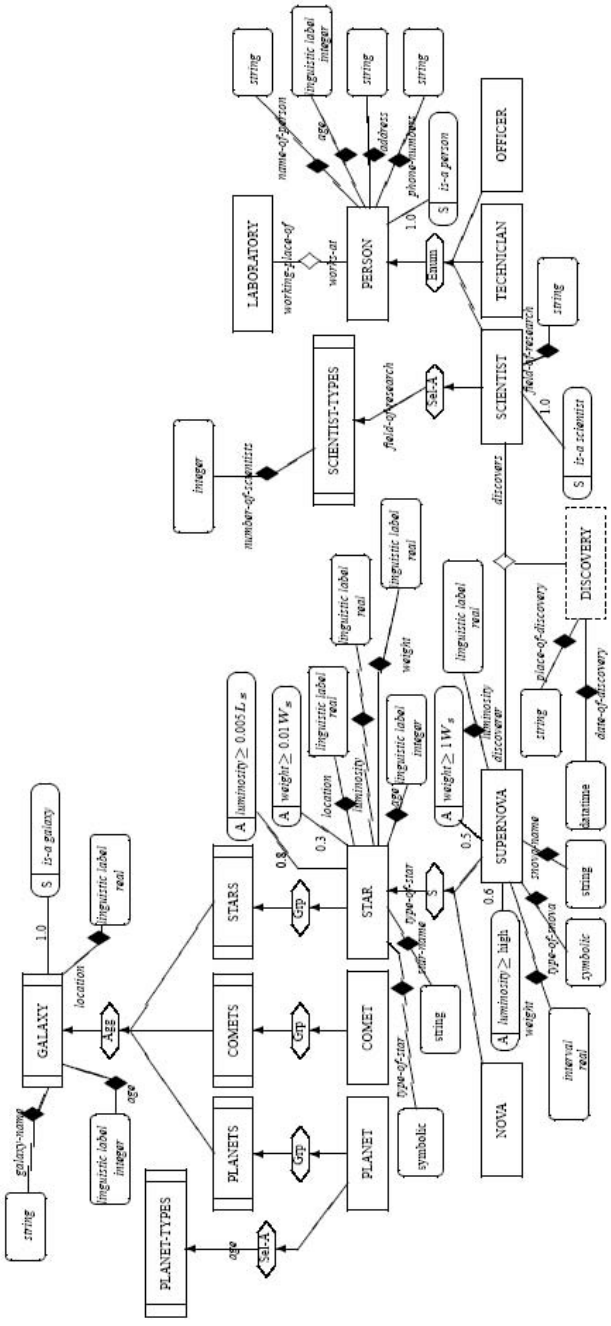

Fig. 1. Example of a FSM-based model

means that this entity verifies fully the property. The number v_i is the value of the attribute of entity e on which the property p_i is defined. For extent properties based on common semantics, v_i is a semantic phrase and the partial d.o.m $\rho_{P_K^i}(v_i)$ is supposed to be equal to 1 but the user may explicitly provide a value less than 1. More generally, the value of $\rho_{P_K^i}(v_i)$ represents the extent to which entity e verifies property p_i of fuzzy class K. Thus, the *global* d.o.m of the fuzzy entity e in the fuzzy class K is:

$$\mu_K(e) = \frac{\sum_{i=1}^{n} \rho_{P_K^i}(v_i) \cdot w_i}{\sum_{i=1}^{n} w_i}. \quad (1)$$

In FSM each fuzzy class is uniquely identified with a name. Each class has a list of characteristics or properties, called attributes. Some of these attributes are used to construct the extent set X_K defined above. To be a member of a fuzzy class K, a fuzzy entity e must verify (fully or partially) at least one of the extent properties, i.e., $\mu_K(e) > 0$. The classes in FSM are categorized as exact or fuzzy. An *exact class* K is a class that all its members have a d.o.m equal to 1. A *fuzzy class* K is a class that at least one of its members has a d.o.m strictly inferior to 1.

The elements of a fuzzy class are called *members*. In FSM, α-MEMBERS denotes for a given fuzzy class K the set $\{e : e \in K \wedge \mu_K(e) \geq \alpha\}$; where $\alpha \in [0, 1]$. It is easy to see that α-MEMBERS $\subseteq$ β-MEMBERS for all α and β in [0,1] and verifying $\alpha \geq \beta$. Note that 1-MEMBERS may also be refereed to *true* or *exact members*. In turn, α-MEMBERS with $0 < \alpha < 1$ are called *fuzzy members*.

FSM supports four different relationships: property, decision-rule, membering and interaction. The *property relationships* relate fuzzy classes to domain classes. Each property relationship creates an attribute. The *decision rule relationships* are an implementation of the extents of fuzzy classes, i.e., the set of properties-based rules used to assign fuzzy entities to fuzzy classes. The *membering relationships* relate fuzzy entities to fuzzy classes through the definition of their d.o.m. The *interaction relationships* relate members of one fuzzy class to other members of one or several fuzzy classes.

In FSM there are several complex fuzzy classes, that permit to implement the semantics of real-world among objects in terms of generalization, specialization, aggregation, grouping and composition relationships, which are commonly used in purely semantic modelling.

To close this section, we provide in Figure 1 a database example that illustrates most of FSM constructs. It will be used for illustration. In the example database, GALAXY is an aggregate fuzzy class whose members are unique collections of members from COMETS, STARS and PLANETS fuzzy grouping classes. These last ones are homogenous collections of members from strong fuzzy classes COMET, STAR and PLANET, respectively. NOVA and SUPERNOVA are two attribute-defined fuzzy subclasses of STAR basing on *type-of-star* attribute. PLANET-TYPES is an attribute-defined fuzzy composite class. This composition is from PLANET fuzzy class basing on the *age* attribute. PERSON is an exact class. It has three enumerated subclasses: SCIENTIST,

TECHNICIAN and OFFICER. Each person is affiliated with at least one LABORATORY. SCIENTIST is a collection of scientists and DISCOVERY is an interaction fuzzy class between SUPERNOVA and SCIENTIST. SCIENTIST-TYPES is a fuzzy composite class from SCIENTIST basing on *field-of-research* attribute.

3 Implementation issues

3.1 Representing imperfect information

FRO supportes a rich set of imperfect data types (see [1] for details). First, it is important to mention that for facilitating data manipulation and for computing efficiency while giving the maximum flexibility to the users, the different types of attributes values in FRO are uniformly represented through possibility distribution. Each of these data types has one, two, three or four parameters permitting to generate its possibility distribution. For example, the graphical representation of possibility distribution of the *fuzzy range* data that handles the "more or less" information is provided in Figure 2. For instance, we may have: "*age* = more or less between 20 and 30". The d.o.m of any z in the fuzzy set A on which the attribute is defined is computed through Equation (2):

$$\mu_A(z) = \begin{cases} 1, & \texttt{if } \beta \leq z \leq \gamma; \\ \frac{\lambda - z}{\lambda - \gamma}, & \texttt{if } \gamma < z < \lambda; \\ \frac{z - \alpha}{\beta - \alpha}, & \texttt{if } \alpha < z < \beta; \\ 0, & \texttt{otherwise.} \end{cases} \tag{2}$$

The parameters β and γ represent the support of the fuzzy set associated with the attribute values and α and λ represent the limits of the transition zones.

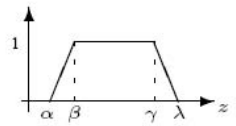

Fig. 2. Possibility distribution of "fuzzy range" data type

3.2 Implementing imperfect information

Several meta-relations have been defined to implement FRO. For example, to store the specificity of all the attributes, we define a meta-relation, called ATTRIBUTES with the following attributes: (i) *attr-id*: it uniquely identifies each attribute; (ii) *attr-name*: it stores the name of the attribute; (iii) *class-name*: denotes the fuzzy class to which the attribute belongs; and (iv) *data-type*: which is a multi-valued attribute that stores the attribute type. For crisp attributes, this

attribute works as in conventional databases (it may take the values of integer, float, etc.). For fuzzy attributes, the *data-type* attribute stores the fuzzy data type itself and the basic crisp data type on which the fuzzy data type is based. The ATTRIBUTES meta-relation associated with the model in Figure 1 is as follows:

attr-id	*attr-name*	*class-name*	*data-type*
attr-15	*star-name*	STAR	{string}
attr-20	*weight*	STAR	{interval, real}

At the extent definition of fuzzy classes, each attribute is mapped into a new composite with three component attributes: (i) *attr-value*: stores the value of the attribute as provided by the user; (ii) *data-type*: stores the data type of the value being inserted; and (iii) *parameters*: is a multi-valued attribute used to store parameters associated with the attribute value. The *data-type* attribute is used both at the extent definition and in the intent definition to allow users insert values of different data types, which may have different number of parameters. The mapping of the *weight* attribute of fuzzy class STAR is as follows:

weight
attr-value|*data-type*|*parameters*
{10W_s, real, {nil}}
{about 17W_s, approximate value, {15,17,18}}

3.3 Mapping of a FSM-based model

As mentioned above, FSM-based model is mapped into a fuzzy relational object (FRO) database one. The FRO was implemented as a front-ends of the relational object database system PostgreSQL. Here we provide the transformation of only fuzzy subclass/superclass relationships. A fuzzy subclass B of a fuzzy superclass A is mapped into a relation which inherits all the attributes of the relation transformed from A. In addition to the attribute *dom*, the relation B contains a new attribute, denoted by *dom-A*, which is used to store the d.o.m of one entity from fuzzy subclass B in its fuzzy superclass A. The same reasoning is used for fuzzy subclasses with more than one fuzzy superclass. Note particulary that the relation mapped from fuzzy class B will contain several *dom-A*, one for each fuzzy superclass. For instance, the mapping of the fuzzy subclass SUPERNOVA in Figure 1 is as follows:

snova-name	*type-of-snova*	$\cdots$	*dom*	*dom-star*
SN1987a	IIb	$\cdots$	0.95	1.0
SN1006	Unknown	$\cdots$	0.7	0.9

3.4 Computing the d.o.m

In FSM, an attribute-based extent property is associated with a condition of the form: <`left-hand-operand`> <`op`> <`right-hand-operand`>. The left-side parameter indicates the attribute name on which the rule is based. The right-side parameter may be a crisp or fuzzy value. The parameter *op* is a binary or a set

operator. For instance, we may have the following decision rules: *luminosity* $\geq$ `high`; and *age* $\in$ `[17-21]`. These operators may be associated with the negation operator "not". Basing on the work of [6], we have extended all the operators that may be used in the definition of the extent properties of fuzzy classes and for query processing. The extended operators apply both for crisp and imprecise data. In this second case, Zadeh's extension principle is used. For instance, the fuzzy "$\simeq$" that gives the degree in which two fuzzy numbers are approximately equal is computed as in Equation (3):

$$\mu_{\simeq}(\tilde{x}, \tilde{y}) = \begin{cases} 0, & \mid \tilde{x} - \tilde{y} \mid > \mathtt{m}; \\ 1 - \frac{\mid \tilde{x} - \tilde{y} \mid}{\mathtt{m}}, & \mid \tilde{x} - \tilde{y} \mid \leq \mathtt{m}. \end{cases} \tag{3}$$

The parameter `m` represents the common support of fuzzy numbers $\tilde{x}$ and $\tilde{y}$. The proposed calculus delivers a index of possibility and the computed degrees are values of possibility obtained through Zadeh's extension principle. A degree of necessity can be also computed.

4 Perspective class and qualification

In this section we introduce the concepts of perspective class and qualification. In the generic definitions below we adopt the following conventions: (i) [ ]: optional parameter(s); (ii){ }: list of parameters or values; (iii) | : the equivalent of the binary operator "xor"; (iv) < > : obligatory parameter(s); and (v) ( ) : series of parameters connected with the "xor" operator. The notion of *perspective class* is introduced in [5]. It is defined as the class with which the user is primarily interested when formulating his/her query. It simplifies query formation and allows users with different interests to approach the database from points of view appropriate to their needs [5]. The perspective class can be associated with an appropriate syntactic process, called *qualification*, allowing immediate attributes of other classes to be treated as if they were attributes of the perspective class. This process may be extended through the entity-valued attributes concept to the attributes related by more than one level of qualification. (The entity-valued attributes are specific, non printable, binary relationships that describe the properties of each entity of a class by relating it to an entity (or entities) of another (or the same) class.) These attributes are called *extended attributes*. For example, in Figure 1, *field-of-research* is an immediate attribute of SCIENTIST and *name-of-person* is an inherited attribute of SCIENTIST from PERSON.

Suppose that classes SCIENTIST and TECHNICIAN in Figure 1 are related and two entity-valued attributes *supervises* (from the point of view of SCIENTIST) and *supervisor* (from the point of view of TECHNICIAN) are defined for them. Then, with TECHNICIAN as perspective class, the *name-of-person* of *supervisor* refers to the name of a technician's supervisor(s) (i.e. a scientist entity). This last qualification is an extended attribute of TECHNICIAN in this example.

Furthermore, the notion of perspective class can be combined with generalization hierarchies to simplify query formation. For example, consider again the

hypothetical binary relationship between SCIENTIST and TECHNICIAN, then the list of technicians and the name of their supervisors can simply be obtained as follows (the syntax of a retrieve query in FSM is provided in §5):

FROM *technician* RETRIEVE *name-of-person*, *name-of-person* OF *supervisor*

In this example TECHNICIAN is the perspective class. This query lists the name of all technician and for each one it provides the name of his/her supervisor but if a technician has no supervisor, whose name should be returned with a null value for the supervisor's name attribute. In this example, the qualification avoided the necessity to put the SCIENTIST class in the FROM clause since the entities of this class are "reached" through the entity-valued relationship.

The general syntax of qualification of an attribute is as follows [5]:

<*attr-name*> ({OF <*entity-valued-attribute-name*> [AS <*class-name*>]})
OF <*perspective-class-name*> [AS <*class-name*>]

The *attr-name* is either a data-valued or an entity-valued attribute. The "AS" clause specifies subclass/superclass role conversion (from a superclass to a subclass) in the same generalization hierarchy and may be best thought of as "looking down" a generalization hierarchy [5]. The following are some examples of qualification from Figure 1:

name-of-person OF *discoverer* OF *supernova*
laboratory-address OF *working-place-of* OF *person*

In the first qualification the perspective class is SUPERNOVA. It returns for each supernova the name(s) of its discoverer(s). The second one uses PERSON as the perspective class. It returns for each person in the database the address of the laboratories he works at.

5 Syntax of retrieve queries

The generic syntax of a retrieve query in FSM is as follows :

[FROM {(<*pers-class-name*> [WITH DOM < op_1 > <*class-level*>]| < α−MEMBERS OF *pers-class-name*>)}]
RETRIEVE <*target-list*>
[ORDER BY <*order-list*>]
[WHERE <*select-expression*> [WITH DOM < op_2 ><*attr-level*>]]

The argument of the FROM statement is a list of perspective classes names (*pers-class-name*) with their respective levels of selection (*class-level*) or a specification of the $\alpha-$MEMBERS to be considered. Only members that have a global d.o.m verifying the arithmetic comparison ensured by the operator op_1 (when the "WITH DOM" part is used) or have a d.o.m greater or equal to α are considered in the selection process. We remark that the WITH DOM part in the FROM clause is facultative and when omitted, all the entities of *perspective-class-name* that verify the WHERE clause are returned. This avoid the necessity of introducing the "WITH DOM > 0" condition when no restriction is imposed on the global d.o.m. of the entities as in queries 4 hereafter. The *target-list* in

the RETRIEVE statement is a list of expressions made up of constants, immediate, inherited and extended attributes of the perspective class, and aggregate and other functions applied on such attributes. The ORDER BY statement is used to choose the way the list of entities is ordered. The *select-expression* in the WHERE statement is a set of symbolic, numerical or logical conditions that should be verified by the attributes of all selected entities. When it is necessary, attributes-based conditions may be combined with appropriate selection levels (*attr-level*) and only entities that their attributes values have a partial d.o.m verifying the arithmetic comparison ensured by the operator op_2 are selected. The following are some illustrative examples of data retrieve operations taken from Figure 1.

Query 1. Retrieve the name and type of supernova that have global d.o.m equal to or greater than 0.7 and have luminosity greater than $15L_s$ with partial d.o.m equal to or greater than 0.9. The symbol L_s is the luminosity of the sun, often used as measurement unit.

FROM *supernova* WITH DOM $\geq$ 0.7 RETRIEVE *snova-name*, *type-of-snova* WHERE *luminosity*$>$ 15L_s WITH DOM $\geq$ 0.9

Query 2. Retrieve the name of all true supernovae and the names of their discoverers.

FROM 1-MEMBERS OF *supernova* RETRIEVE *snova-name*, *name-of-person* OF *discoverer* OF *supernova*

Here the qualification permits to avoid the necessity of adding the class SCIENTIST in the FROM clause. In addition, it avoids the necessity of a WHERE clause.

Query 3. Retrieve dates of discoveries and names of all supernovae of type "Ia" that are located in milky-way galaxy with a global d.o.m greater than 0.5 and having high luminosity with d.o.m less than 0.7.

FROM *discovery*, *supernova*, *galaxy* RETRIEVE *snova-name*, *date-of-discovery*
WHERE *type-of-snova* = "Ia" and (*galaxy-name*="milky-way" and *galaxy.location* = *supernova.location*
WITH DOM $>$ 0.5) and *luminosity*= high WITH DOM $<$ 0.7

In this query example as in the next one, several perspective classes are used. In addition, the "WITH DOM" part is omitted from the FROM clause and so the conditions of the WHERE clause will be checked for all the entities.

Query 4. Retrieve the name, the date of discovery and the discoverer of all supernovae which are not located in the milky-way galaxy with d.o.m not less than 0.5.

FROM *supernova*, *discovery*
RETRIEVE *snova-name*, *date-of-discovery*, *name-of-person* OF *discoverer* OF *supernova*
WHERE *supernova.location* not in (FROM *galaxy* RETRIEVE *location* WHERE *galaxy-name*="milky-way")
WITH DOM $\geq$ 0.5

This example illustrates an imbricated query in FSM.

6 Query processing

The query processing schema (under implementation) contains three phases:

– **Phase 1**: Syntactic analysis.

– **Phase 2**: Verification of the conditions specified in the `FROM` statement. It returns, for each tuple, a *global satisfaction degree* $d_g \in [0,1]$ measuring the level to which the tuple satisfies the *class-level conditions*. The tuples for which $d_p > 0$ represent the input for the next phase.
– **Phase 3**: Associate to each tuple a *partial satisfaction degree* $d_p \in [0,1]$ measuring the level to which tuples satisfy the *entity-level conditions*.

The overall satisfaction $d_o \in [0,1]$ is computed as $d_o = d_g * d_p$.

The conditions in the `WHERE` statement may be connected with `AND`, `A OR` or `NOT`. In this case, the computing of partial satisfaction degrees $d_p > 0$ are as follows (for two conditions c_1 and c_2):

Connector	$d_p(c_1)$	$d_p(c_2)$	d_p
AND	α	β	$\alpha * \beta$ or $\min\{\alpha, \beta\}$
OR	α	β	$\max\{\alpha, \beta\}$
NOT	α	–	$1 - \alpha$

where: $d_p(c_2)$ and $d_p(c_2)$ is the partial satisfaction degrees for c_1 and c_2, respectively; and α and β are in $[0,1]$.

7 Conclusion

In this paper, we have introduced an ongoing query language devoted to FSM-based databases. This query language uses the notions of perspective class and qualification. These concepts permits to simplify query formation and allows users with different interests to approach the database from points of view appropriate to their needs. When combined with fuzzy set theory, they provide a simplified and powerful query language. The implementation of this proposed language is ongoing.

References

1. Bahri, A., Bouaziz, R., Chakhar, S. Naïja, Y.: Implementing imperfect information in fuzzy databases. In: Proc. SCIII'2005. Tunis, Tunisia, October 14-16, 2005
2. Bosc, P., Kraft, D., Petry, F.: Fuzzy sets in database and information systems: Status and opportunities, Fuz. Sets and Sys. **156** (2005) 418–426
3. Bouaziz, R. Chakhar, S., Mousseau, V., Sudah, R., Telmoudi, A.: Database design, implementation and querying within the fuzzy semantic model. Inf. Sci. (to appear)
4. Chakhar, S., Telmoudi, A.: Conceptual design and implementation of the fuzzy semantic model, In: Proc. IPMU'2006. Paris, France, July 2-7, (2006) pp. 2438-2445
5. Fritchman, B.L., Guck, R.L., Jagannathan, D., Thompson, J.P., Tolbret, D.M.: SIM: Design and implementation of a semantic database system. In: ACM SIGMOD Conf. (1989) 46–55
6. Medina, J.M., Vila, M.A., Cubero, J.C., Pons, O.: Towards the implementation of a generalized fuzzy relational database model. Fuz. Sets and Sys. **75** (1995) 273–289
7. Ma, Z.M., Zhang, W.J., Ma, W.Y.: Extending object-oriented databases for fuzzy information modeling. Inf. Sys. **29** (2004) 421–435
8. Ribeiro, R.A., Moreira, A.M.: Fuzzy query interface for a business database. Int. J. of Hum.-Comp. Stu. **58** (2003) 363–391